ZigBee2007 协议栈 API 函数使用说明

第一章 介绍

这份文档为目前发布的 ZigBee 2007 协议栈提供了应用程序接口函数（API）的使用说明。为使得我们更好地开发和理解 ZigBee 项目，这份文档详细的讲述了协议栈中的数据结构和函数调用。首先，我们来了解一下在 ZigBee 2007 协议栈中使用的各个层次：

- ZDO

  设备对象层，即 ZDO（ZigBee Device Object）层，提供了管理一个 ZigBee 节点所要使用的功能函数。ZDO API 为协调器、路由器和终端设备提供了应用端点的管理函数，其中包括：建立、发现和加入一个 ZigBee 网络，绑定应用端点和安全管理。

- AF

  应用框架层，即 AF（Application Framework），提供了针对协议栈的应用端点（EndPoint1~240）和设备对象端点（EndPoint0）接口，其中主要包含：设备描述数据结构和数据收、发函数。

- APS

  应用支持子层，即 APS（Application Support Sublayer），为设备对象和应用实体提供了一系列的支持服务。

- NWK

  网络层，即 NWK（ZigBee network），为上层提供了管理服务和数据服务。

- ZMAC

  介质访问层，即 ZMAC，在 802.15.4 MAC 与 网络层之间提供接口。

第二章 应用函数接口（API）

接下来我们讲述的应用函数接口主要包含一些经常使用的数据结构和各层提供的关键性函数。

2.1 设备对象（ZDO）

本节列举出了在 ZigBee 设备规范，即 ZigBee Device Profile（ZDP），所定义的相关命令和响应函数。

2.1.1 概述

ZDP 描述了 ZDO 内部一般性的 ZigBee 设备功能是如何实现的。它定义了使用命令和响应对的设备描述和簇。ZDP 为 ZDO 和应用程序提供如下功能：

- 设备网络启动
- 设备和服务发现
- 终端设备绑定、绑定和取消绑定服务
- 网络管理服务

2.1.2 ZDO 网络设备启动

通过默认的 ZDApp_Init()（在 ZDApp.c 中）启动 ZigBee 网络中的设备。但是一个应用程序可以跳过这个默认行为：

ZDApp.c

```
#if defined( HOLD_AUTO_START )
  devStates_t devState = DEV_HOLD;
#else
  devStates_t devState = DEV_INIT;
#endif
```

其中 HOLD_AUTO_START 在 IAR ->Project->Option->C/C++ Compile->Preprocess->Defined symbols

中定义。如果预编译选择 HOLD_AUTO_START，则 ZDO 不启动网络设备，只是闪烁 LED4。交由应用程序启动网络设备。

ZDApp.c

```
void ZDApp_Init( uint8 task_id )
{
    if ( devState != DEV_HOLD )
    {
      ZDOInitDevice( 0 );
    }
    else
    {
      // Blink LED to indicate HOLD_START
      HalLedBlink ( HAL_LED_4, 0, 50, 500 );
    }
}
```

sapi.c

```
UINT16 SAPI_ProcessEvent( byte task_id, UINT16 events )
{

  if ( events & ZB_ENTRY_EVENT )
  {
    if ( startOptions & ZCD_STARTOPT_AUTO_START )
    {
      zb_StartRequest();//即调用 ZDOInitDevice (zgStartDelay)
    }
    else
    {
      // blink leds and wait for external input to config and restart
      HalLedBlink(HAL_LED_2, 0, 50, 500);
    }
  }
}
```

ZDOInitDevice()具体描述如下：

函数原型：

uint8 ZDOInitDevice(uint16 startDelay);

参数：

startDelay –设备启动延时（毫秒）

返回值:

ZDO_INITDEV_RESTORED_NETWORK_STATE（网络状态为恢复）

ZDO_INITDEV_NEW_NETWORK_STATE （网络状态为初始化）

ZDO_INITDEV_LEAVE_NOT_STARTED （网络状态为未启动） 举例

分析:

2.1.3 ZDO 信息回调函数

通过函数 ZDO_RegisterForZDOMsg()注册请求或响应消息，这样就可以接收其他设备无线传输的消息。

2.1.3 .1 ZDO_RegisterForZDOMsg()

调用该函数可以将接收到的无线传输信息复制一份到 OSAL 层的某个任务。该任务接收到消息后可以自行解析此消息或者通过 ZDO 解析函数解析此消息。只有响应消息需要使用 ZDO 解析函数。

函数原型:

ZStatus_t ZDO_RegisterForZDOMsg(uint8 taskID, uint16 clusterID);

参数:

taskID -任务ID，该任务发送OSAL 消息;

clustered-簇 ID (例如: NWK_addr_rsp)，在 ZDProfile.h 定义了相关的簇.

返回值:

ZStatus_t-状态

消息接收到之后作为 ZDO_CB_Msg（一种系统消息）发送至应用或者任务,消息结构体 zdoIncomingMsg_t 定义在 ZDProfile.h 中：

```
#define ZDO_CB_MSG                    0xD3      // ZDO incoming message callback
```

消息结构体 zdoIncomingMsg_t:

```
typedef struct
{
  osal_event_hdr_t hdr;
  zAddrType_t      srcAddr;
  uint8            wasBroadcast;
  cId_t            clusterID; uint8
                   SecurityUse;
  uint8            TransSeq;
  uint8            asduLen;
  uint16           macDestAddr;
  uint8            *asdu;
} zdoIncomingMsg_t;
```

在应用层使用该函数注册的消息有：

```
void SAPI_Init( byte task_id )
{
  // Register callback evetns from the ZDApp
  ZDO_RegisterForZDOMsg( sapi_TaskID, NWK_addr_rsp );
  ZDO_RegisterForZDOMsg( sapi_TaskID, Match_Desc_rsp );
}
```

调用 ZDO_RegisterForZDOMsg()在应用层注册后，应用层处理接收到的消息方式如下：

```
UINT16 SAPI_ProcessEvent( byte task_id, UINT16 events )
{
  osal_event_hdr_t *pMsg;

  if ( events & SYS_EVENT_MSG )
  {
    pMsg = (osal_event_hdr_t *) osal_msg_receive( task_id );
    while ( pMsg )
    {
      switch ( pMsg->event )
      {
        case ZDO_CB_MSG:
          SAPI_ProcessZDOMsgs( (zdoIncomingMsg_t *)pMsg );
          break;
      }
    }
```

```
  }
}
```

在 ZDO 层使用该函数注册的消息有:

```
void ZDApp_RegisterCBs( void )
{
#if defined ( ZDO_IEEEADDR_REQUEST ) || defined ( REFLECTOR )
  ZDO_RegisterForZDOMsg( ZDAppTaskID, IEEE_addr_rsp );
#endif
#if defined ( ZDO_NWKADDR_REQUEST ) || defined ( REFLECTOR )
  ZDO_RegisterForZDOMsg( ZDAppTaskID, NWK_addr_rsp );
#endif
#if ZG_BUILD_COORDINATOR_TYPE
  ZDO_RegisterForZDOMsg( ZDAppTaskID, Bind_rsp );
  ZDO_RegisterForZDOMsg( ZDAppTaskID, Unbind_rsp );
  ZDO_RegisterForZDOMsg( ZDAppTaskID, End_Device_Bind_req );
#endif
#if defined ( REFLECTOR )
  ZDO_RegisterForZDOMsg( ZDAppTaskID, Bind_req );
  ZDO_RegisterForZDOMsg( ZDAppTaskID, Unbind_req );
#endif
}
```

调用 ZDO_RegisterForZDOMsg()在 ZDO 层注册后，ZDO 层处理接收到的消息方式如下:

```
void ZDApp_ProcessOSALMsg( osal_event_hdr_t *msgPtr )
{
  switch ( msgPtr->event )
  {
    case ZDO_CB_MSG:
      ZDApp_ProcessMsgCBs( (zdoIncomingMsg_t *)msgPtr );
      break;
  }
}
```

2.1.3 .2 ZDO_RemoveRegistedCB()

调用此函数取消请求无线传输的消息。

函数原型:

ZStatus_t ZDO_RemoveRegistedCB (uint8 taskID, uint16 clusterID);

参数：

taskID -任务ID，该任务ID必须与ZDO_RegisterForZDOMsg()所注册的任务ID相同；

clustered-簇 ID，该簇 ID 必须与 ZDO_RegisterForZDOMsg()所使用的簇 ID 相同。

返回值：

ZStatus_t-状态

2.1.4 ZDO 发现 API

ZDO 发现 API 包含建立和发送 ZDO 设备和服务发现请求和响应。所有这些 API 函数和 ZDP 命令（ZigBee Device Profile Command）如下表：

ZDP_NwkAddrReq()	NWK_addr_req
ZDP_NWKAddrRsp()	NWK_addr_rsp
ZDP_IEEEAddrReq()	IEEE_addr_req
ZDP_IEEEAddrRsp()	IEEE_addr_rsp
ZDP_NodeDescReq()	Node_Desc_req
ZDP_NodeDescRsp()	Node_Desc_rsp
ZDP_PowerDescReq()	Power_Desc_req
ZDP_PowerDescRsp()	Power_Desc_rsp
ZDP_SimpleDescReq()	Simple_Desc_req
ZDP_SimpleDescRsp()	Simple_Desc_rsp
ZDP_ComplexDescReq()	Complex_Desc_req
ZDP_ActiveEPIFReq()	Active_EP_req
ZDP_ActiveEPIFRsp()	Active_EP_rsp

ZDP_MatchDescReq()	Match_Desc_req
ZDP_MatchDescRsp()	Match_Desc_rsp
ZDP_UserDescSet()	User_Desc_set
ZDP_UserDescConf()	User_Desc_conf
ZDP_UserDescReq()	User_Desc_req
ZDP_UserDescRsp()	User_Desc_rsp
ZDP_DeviceAnnce()	Device_annce
ZDP_ServerDiscReq()	System_Server_Discovery_req
ZDP_ServerDiscRsp()	System_Server_Discovery_rsp

2.1.4.1 ZDP_NwkAddrReq()

调用此函数将生成一个根据已知 IEEE 地址询问远程节点 16 位网络地址的消息。这个消息作为一个广播消息发送至网络中的所有节点。

函数原型：

afStatus_t ZDP_NwkAddrReq(byte *IEEEAddress, byte ReqType, byte StartIndex, byte SecuritySuite);

参数：

IEEEAddress-远程节点的IEEE地址

ReqType- ZDP_NWKADDR_REQTYPE_SINGLE（只返回节点的短地址和扩展地址）、ZDP_NWKADDR_REQTYPE_EXTENDED（返回节点的短地址和扩展地址以及所有相关节点的短地址）

StartIndex-响应节点的响应信息可以有很多的响应选项，请求程序可以指定一个起始索引号，索引从0开始

SecuritySuite –安全要求

返回值:

ZStatus_t-状态

2.1.4.2 ZDP_NWKAddrRsp()

ZDP_NWKAddrRsp()实际上是调用ZDP_ADDRRsp()这个宏定义,用于建立和发送16位网络地址响应。

函数原型:

afStatus_t ZDP_NWKAddrRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status,

byte *IEEEAddrRemoteDev, byte ReqType, uint16 nwkAddr, byte NumAssocDev, byte StartIndex,

uint16 *NWKAddrAssocDevList, byte SecuritySuite);

参数:

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0，ZDP_INVALID_REQTYPE=1，ZDP_DEVICE_NOT_FOUND=2

IEEEAddrRemoteDev –远程节点的64位IEEE地址

ReqType –请求的类型

nwkAddr – 远程节点的16位网络地址

NumAssocDev –与远程节点关联的节点数目

StartIndex –响应节点的响应信息可以有很多的响应选项，请求程序可以指定一个起始索引号，该索

引号是响应信息的起始索引号

NWKAddrAssocDevList – 与远程节点关联的节点16位网络地址列表

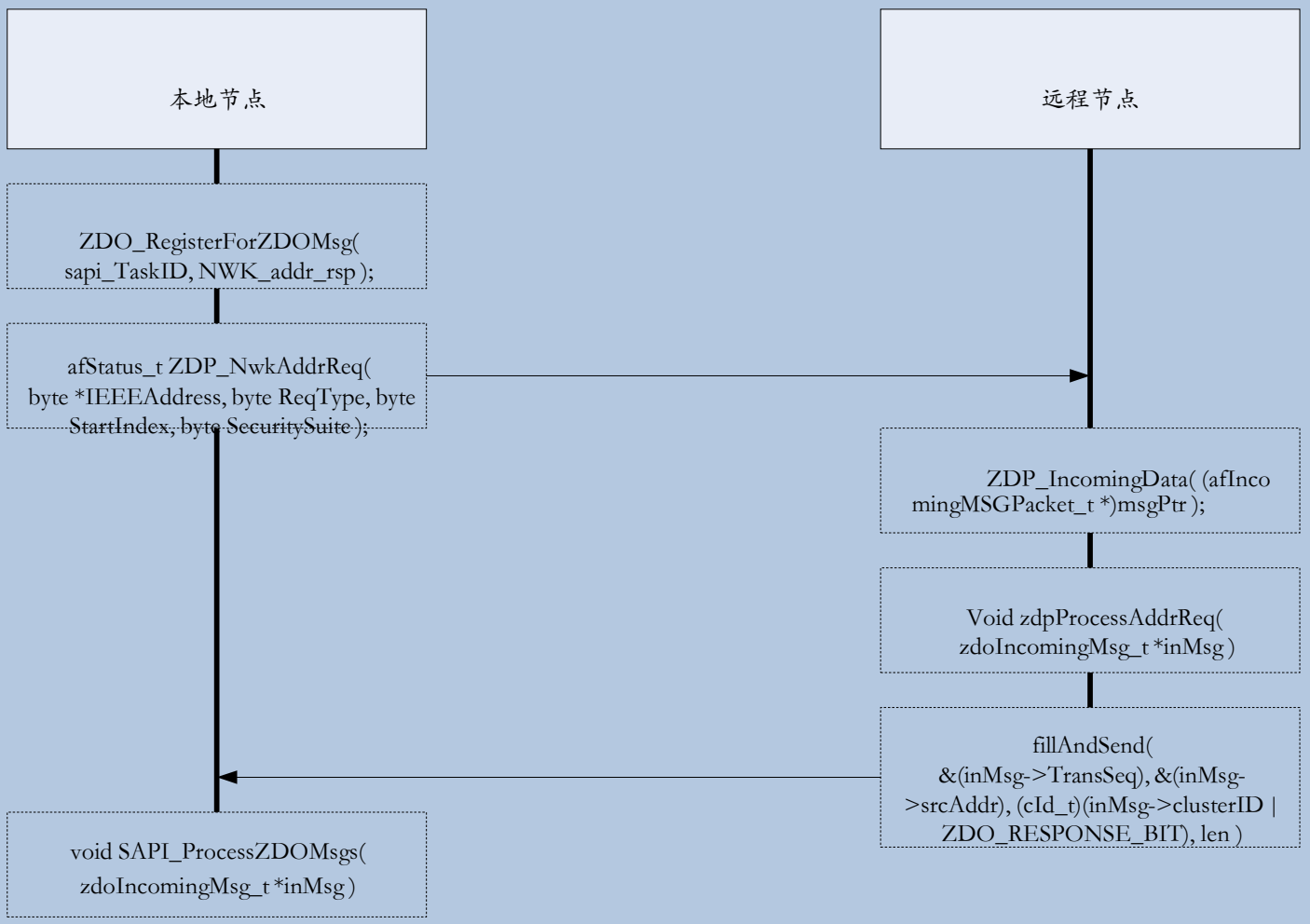
SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

2.1.4.3 网络地址请求和响应应用举例分析

本地节点首先需要在应用层或 ZDO 层注册响应信息，本例中在 sapi.c 中注册了 NWK_addr_rsp 信息。然后调用 ZDP_NwkAddrReq()请求远程节点的网络地址。远程节点接收到该请求信息（该信息从属于 AF_DATA_CONFIRM_CMD），则根据 Cluster ID 选择处理函数，本例中使用 zdpProcessAddrReq()来处理网络地址请求。当处理完之后，通过调用 fillAndSend()将响应信息发送至本地节点。由于在应用层中注册了该响应信息，因此调用 SAPI_ProcessZDOMsgs()来处理响应信息。详细流程如下图所示:



2.1.4.4 ZDP_IEEEAddrReq()

调用此函数将生成一个根据已知 16 位网络地址询问远程节点 64 位 IEEE 地址的消息。这个消息直接单播发送至该远程节点。

函数原型:

afStatus_t ZDP_IEEEAddrReq(uint16 shortAddr, byte ReqType, byte StartIndex, byte SecuritySuite);

参数:

shortAddr-远程节点的16位网络地址

ReqType- ZDP_NWKADDR_REQTYPE_SINGLE（只返回节点的短地址和扩展地址）、

ZDP_NWKADDR_REQTYPE_EXTENDED（返回节点的短地址和扩展地址以及所有相关节点的短地

址）

StartIndex-响应节点的响应信息可以有很多的响应选项，请求程序可以指定一个起始索引号，索引从0开始

SecuritySuite –安全要求

返回值：

ZStatus_t-状态

2.1.4.5 ZDP_IEEEAddrRsp()

ZDP_IEEEAddrRsp()实际上是调用ZDP_ADDRRsp()这个宏定义，用于建立和发送64位IEEE地址响应。

函数原型：

afStatus_t ZDP_IEEEAddrRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status,

byte *IEEEAddrRemoteDev, byte ReqType, uint16 nwkAddr, byte NumAssocDev, byte StartIndex,

uint16 *NWKAddrAssocDevList, byte SecuritySuite);

参数：

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0，ZDP_INVALID_REQTYPE=1，ZDP_DEVICE_NOT_FOUND=2

IEEEAddrRemoteDev –远程节点的64位IEEE地址

ReqType –请求的类型

nwkAddr – 远程节点的16位网络地址

NumAssocDev –与远程节点关联的节点数目

StartIndex –响应节点的响应信息可以有很多的响应选项，请求程序可以指定一个起始索引号，该索引号是响应信息的起始索引号

NWKAddrAssocDevList – 与远程节点关联的节点16位网络地址列表

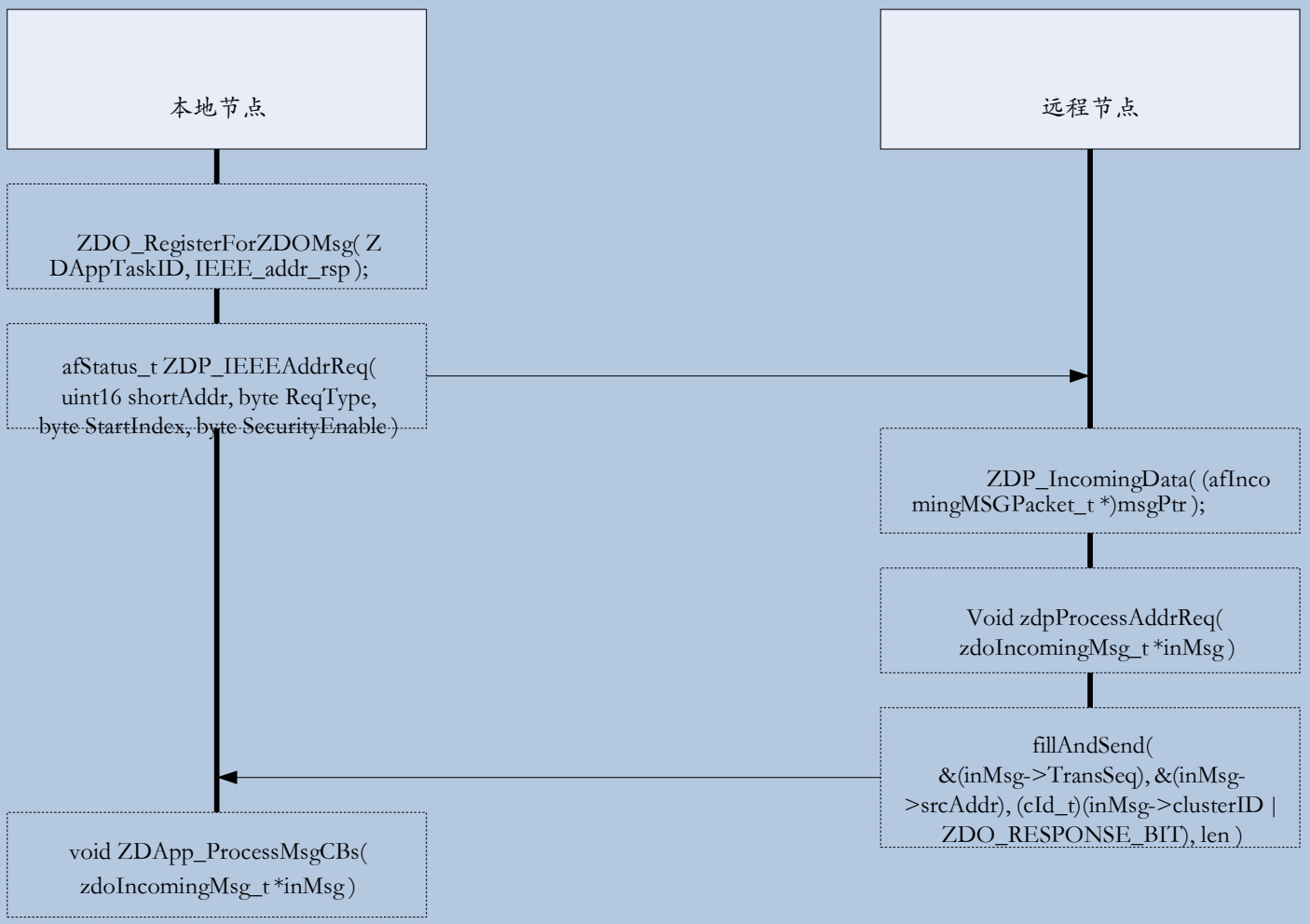
SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

2.1.4.6 IEEE 地址请求和响应应用举例分析

本地节点首先需要在应用层或ZDO层注册响应信息,本例中在ZDApp.c中注册了IEEE_addr_rsp信息。然后调用 ZDP_IEEEAddrReq()请求远程节点的网络地址。远程节点接收到该请求信息（该信息从属于 AF_DATA_CONFIRM_CMD），则根据 Cluster ID 选择处理函数，本例中使用zdpProcessAddrReq()来处理 IEEE 地址请求。当处理完之后，通过调用 fillAndSend()将响应信息发送至本地节点。由于在 ZDO 中注册了该响应信息，因此调用 ZDApp_ProcessMsgCBs()来处理响应信息。详细流程如下图所示:



2.1.4.7 ZDP_NodeDescReq()

ZDP_NodeDescReq()实际上是调用宏定义 ZDP_NWKAddrOfInterestReq()。这个函数建立和发送一个节点描述符（Node Descriptor）请求至已明确网络地址的远程节点。

函数原型：

afStatus_t ZDP_NodeDescReq(zAddrType_t *dstAddr, uint16 NWKAddrOfInterest, byte SecuritySuite);

参数：

DstAddr-目的地址

NWKAddrOfInterest -远程节点的16位网络地址

SecuritySuite -安全要求

返回值:

ZStatus_t-状态

2.1.4.8 ZDP_NodeDescMsg()

调用此函数响应节点描述符的请求。

函数原型:

afStatus_t ZDP_NodeDescMsg(byte TransSeq, zAddrType_t *dstAddr, byte Status, uint16 nwkAddr,

NodeDescriptorFormat_t *pNodeDesc, byte SecuritySuite);

参数:

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0， ZDP_DEVICE_NOT_FOUND=1

nwkAddr–已明确的远程节点的16位网络地址

pNodeDesc –节点描述符

SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

2.1.4.9 节点描述符请求和响应应用举例分析

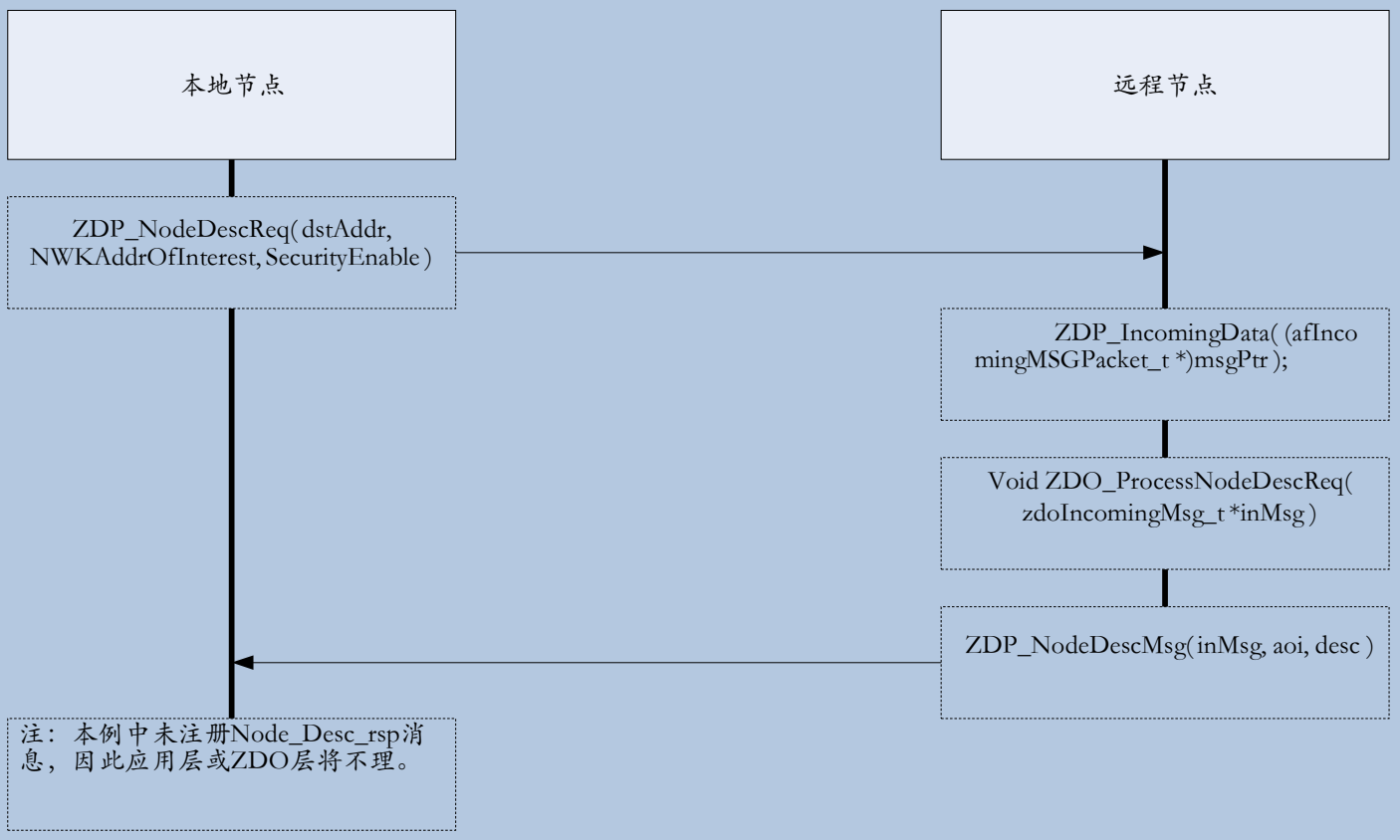
节点描述符包含 ZigBee 节点的功能信息,每个节点只有唯一的一个节点描述符,其数据结构如下:

AF.h

typedef struct	

{ uint8 LogicalType:3; uint8 ComplexDescAvail:1; uint8 UserDescAvail:1; uint8 Reserved:3; uint8 APSFlags:3; uint8 FrequencyBand:5; uint8 CapabilityFlags; uint8 ManufacturerCode[2]; uint8 MaxBufferSize; uint8 MaxInTransferSize[2]; uint16 ServerMask; uint8 MaxOutTransferSize[2]; uint8 DescriptorCapability; } NodeDescriptorFormat_t;	节点逻辑类型。000：协调器；001：路由器；010：终端设备 可否使用复杂描述符。1：可以；0：不可以 可否使用用户描述符。1：可以；0：不可以 保留位 应用支持子层功能标志位。一般为 0 表示不使用 频段位。0：868~868.6MHz；2： 902~928MHz；3： 2400~2483.5MHz 功能标志位。备用协调器、设备类型、电源来源、RXON、安全、分配地址 制造商代码。由 ZigBee 联盟分配 最大缓冲区。网络层数据单元（NSDU）的最大值，以字节为单位 最大输入。应用支持子层数据单元（ASDU）的最大值，以字节为单位 服务器掩码。第 0~6 位可用，其他位保留 最大输出。应用支持子层数据单元（ASDU）的最大值，以字节为单位 描述符能力域。第 0~1 位可用，其他位保留

本地节点在ZDApp.c中调用ZDConfig_UpdateNodeDescriptor()初始化节点描述符，本例中在应用层或ZDO层均未注册Node_Desc_rsp信息。调用ZDP_NodeDescReq()可以根据已明确了的网络地址请求远程节点的节点描述符。远程节点接收到该请求信息（该信息从属于AF_DATA_CONFIRM_CMD），则根据Cluster ID选择处理函数，会使用zdpProcessNodeDescReq()来处理节点描述符请求。当处理完之后，通过调用ZDP_NodeDescMsg(inMsg, aoi, desc)将响应信息发送至本地节点。详细流程如下图所示：



2.1.4.10 ZDP_PowerDescReq()

ZDP_PowerDescReq()实际上是调用宏定义 ZDP_NWKAddrOfInterestReq()。这个函数建立和发送一个电源描述符（Power Descriptor）请求至已明确网络地址的远程节点。

函数原型:

afStatus_t ZDP_PowerDescReq(zAddrType_t *dstAddr, uint16 NWKAddrOfInterest, byte SecuritySuite);

参数:

DstAddr-目的地址

NWKAddrOfInterest -远程节点的16位网络地址

SecuritySuite -安全要求

返回值:

ZStatus_t-状态

2.1.4.11 ZDP_PowerDescMsg()

调用此函数响应电源描述符的请求。

函数原型:

afStatus_t ZDP_PowerDescMsg(byte TransSeq, zAddrType_t *dstAddr, byte Status, uint16 nwkAddr, NodePowerDescriptorFormat_t *pPowerDesc, byte SecuritySuite);

参数:

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0， ZDP_DEVICE_NOT_FOUND=1

nwkAddr–已明确的远程节点的16位网络地址

pPowerDesc –电源描述符

SecuritySuite – 安全要求

返回值:

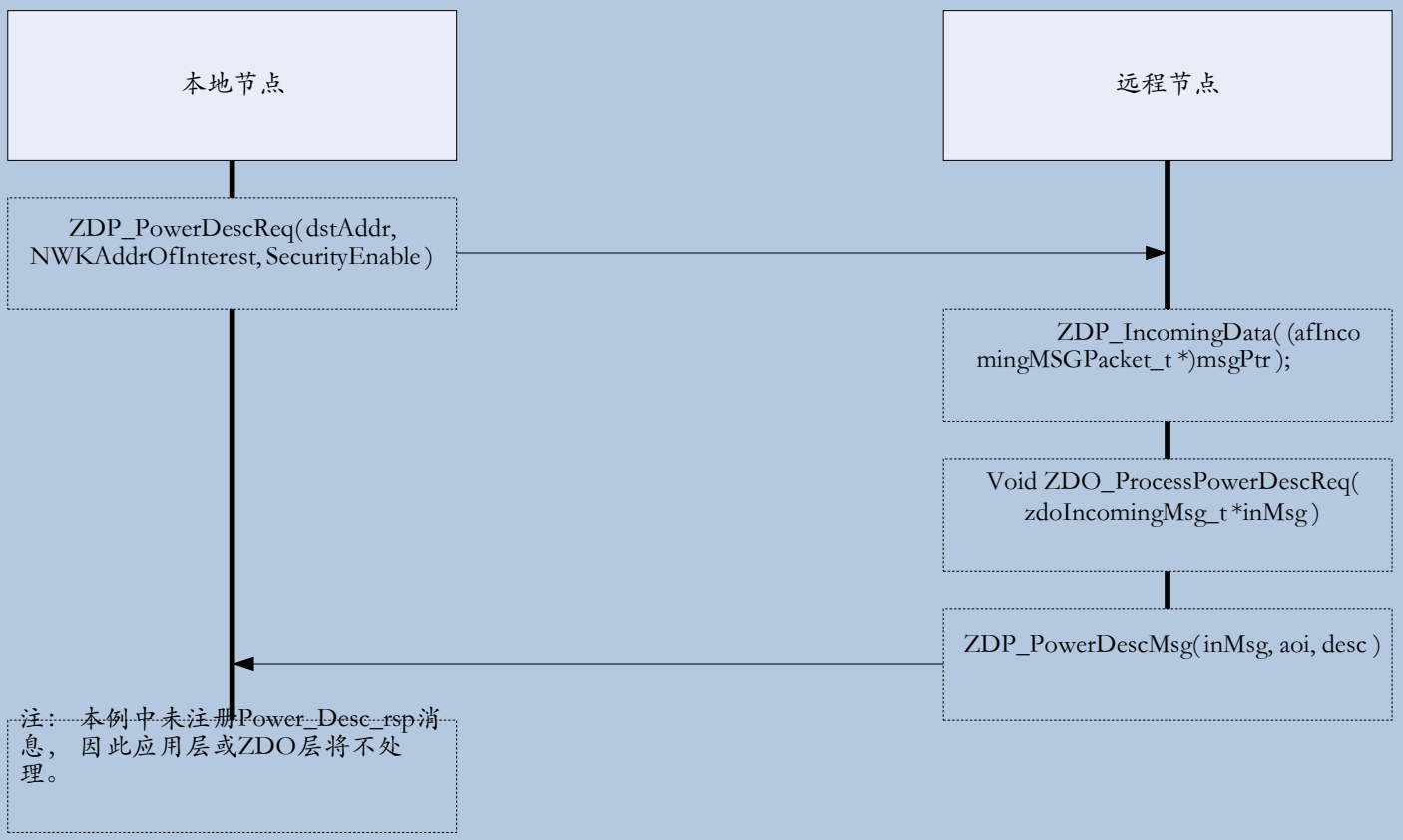
ZStatus_t-状态

2.1.4.12 电源描述符请求和响应应用举例分析

电源描述符动态指示 ZigBee 节点的电源状态信息，每个节点只有唯一的一个电源描述符，其数据结构如下： AF.h

typedef struct { unsigned int PowerMode:4; unsigned int AvailablePowerSources:4; unsigned int CurrentPowerSource:4; unsigned int CurrentPowerSourceLevel:4; } NodePowerDescriptorFormat_t;	电源模式 可用电源来源 当前电源来源 当前电源来源的电量

本地节点在ZDApp.c中调用ZDConfig_UpdatePowerDescriptor()初始化电源描述符，本例中应用层或ZDO层均未注册Power_Desc_rsp信息。调用ZDP_PowerDescReq()可以根据已明确了的网络地址请求远程节点的节点描述符。远程节点接收到该请求信息（该信息从属于AF_DATA_CONFIRM_CMD），则根据Cluster ID选择处理函数，会使用zdpProcessPowerDescReq()来处理电源描述符请求。当处理完之后，通过调用ZDP_PowerDescMsg(inMsg, aoi, desc)将响应信息发送至本地节点。详细流程如下图所示:



2.1.4.13 ZDP_SimpleDescReq()

ZDP_SimpleDescReq()函数建立和发送一个简单描述符（Simple Descriptor）请求至已明确网络地址的远程节点。

函数原型：

afStatus_t ZDP_SimpleDescReq(zAddrType_t *dstAddr, uint16 nwkAddr,byte endPoint, byte SecurityEnable)

参数：

dstAddr - 目的地址

NWKAddr - 远程节点的16位网络地址

Endpoint - 端点号

SecuritySuite -安全要求

返回值:

ZStatus_t-状态

2.1.4.14 ZDP_SimpleDescMsg()

调用此函数响应简单描述符的请求。

函数原型:

afStatus_t ZDP_SimpleDescMsg(byte TransSeq, zAddrType_t *dstAddr, byte Status,

SimpleDescriptorFormat_t *pSimpleDesc, byte SecuritySuite);

参数:

TranSeq-传输序号

DstAddr-目的地址

Status - UCCESS=0， INVALID_EP=1， NOT_ACTIVE=2，DEVICE_NOT_FOUND=3

pSimpleDesc –简单描述符

SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

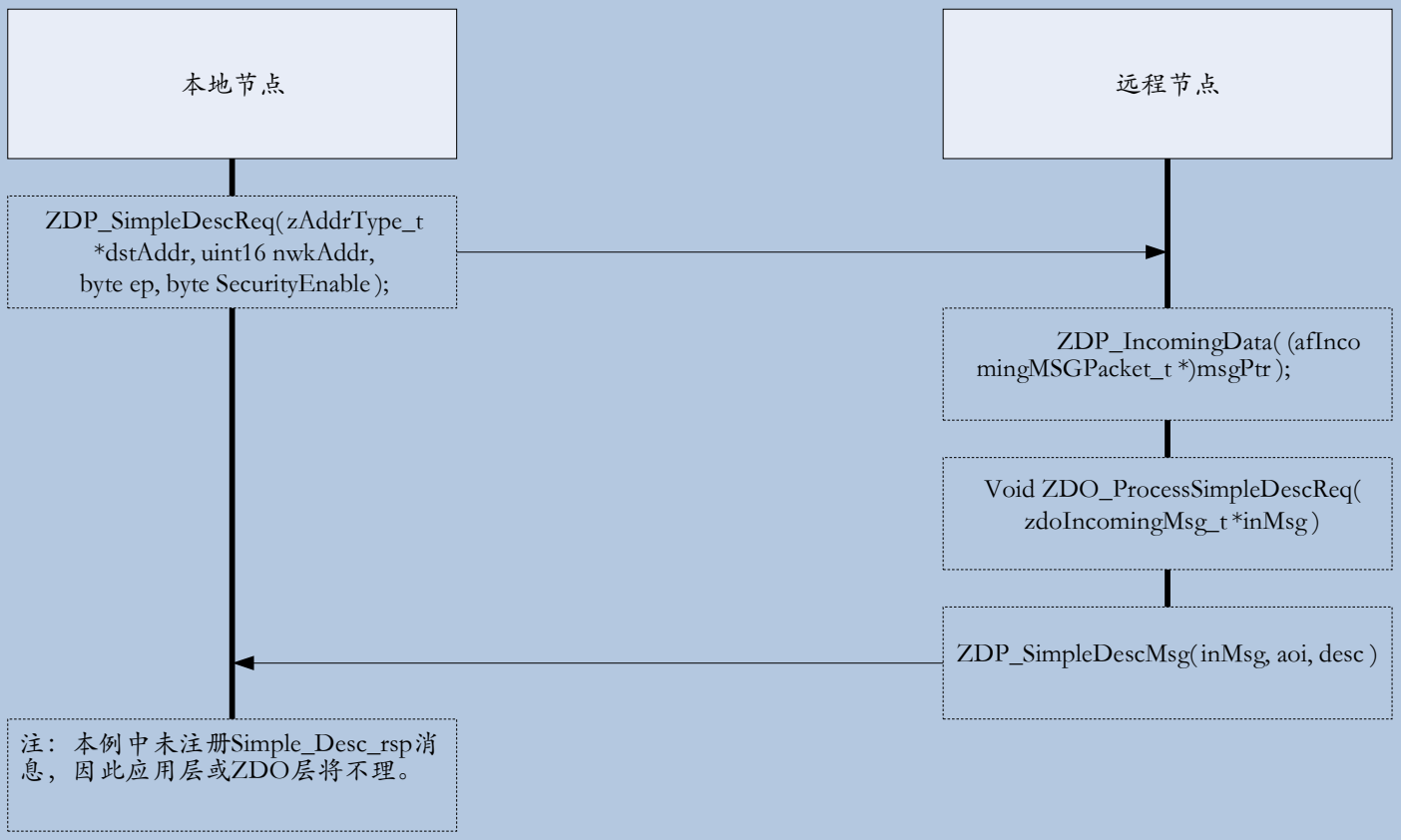
2.1.4.15 简单描述符请求和响应应用举例分析

简单描述符包含该节点所含每个端点的具体信息，其数据结构如下: AF.h

typedef struct {	
byte EndPoint;	端点 应用规范标
uint16 AppProfId;	识符 应用设备标
uint16 AppDeviceId;	识符 应用设备版
byte AppDevVer:4;	本

byte Reserved:4; byte AppNumInClusters; cId_t *pAppInClusterList; byte AppNumOutClusters; cId_t *pAppOutClusterList; } SimpleDescriptionFormat_t;	保留 应用输入簇个数 应用输入簇列表 应用输出簇个数 应用输出簇列表

本例中应用层或ZDO层均未注册Simple_Desc_rsp信息。调用ZDP_SimpleDescReq()可以根据已明确了的网络地址请求远程节点的某个指定端点的简单描述符。远程节点接收到该请求信息（该信息从属于AF_DATA_CONFIRM_CMD），则根据Cluster ID选择处理函数，会使用zdpProcessSimpleDescReq()来处理简单描述符请求。当处理完之后，通过调用ZDP_SimpleDescMsg(inMsg, stat, sDesc)将响应信息发送至本地节点。详细流程如下图所示：



2.1.4.16 ZDP_ActiveEPIFReq ()

ZDP_ActiveEPIFReq ()实际上是调用宏定义 ZDP_NWKAddrOfInterestReq()。这个函数建立和发送一个活动端点请求至已明确网络地址的远程节点。使用这个宏定义请求远程节点所有活动的端点。

函数原型:

afStatus_t ZDP_ActiveEPIFReq(zAddrType_t *dstAddr, uint16 NWKAddrOfInterest, byte SecuritySuite);

参数:

DstAddr-目的地址

NWKAddrOfInterest -远程节点的16位网络地址

SecuritySuite -安全要求

返回值:

ZStatus_t-状态

2.1.4.17 ZDP_ActiveEPIFRsp()

该函数实际上是调用宏定义 ZDP_EPIFRsp().调用此函数响应活动端点请求。

函数原型:

afStatus_t ZDP_ActiveEPIFRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status, uint16 nwkAddr, byte Count, byte *pEPIntfList, byte SecuritySuite);

参数:

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0， ZDP_DEVICE_NOT_FOUND=1

nwkAddr—已明确的远程节点的16位网络地址

Count —活动端点个数

pEPIntfList—活动端点列表

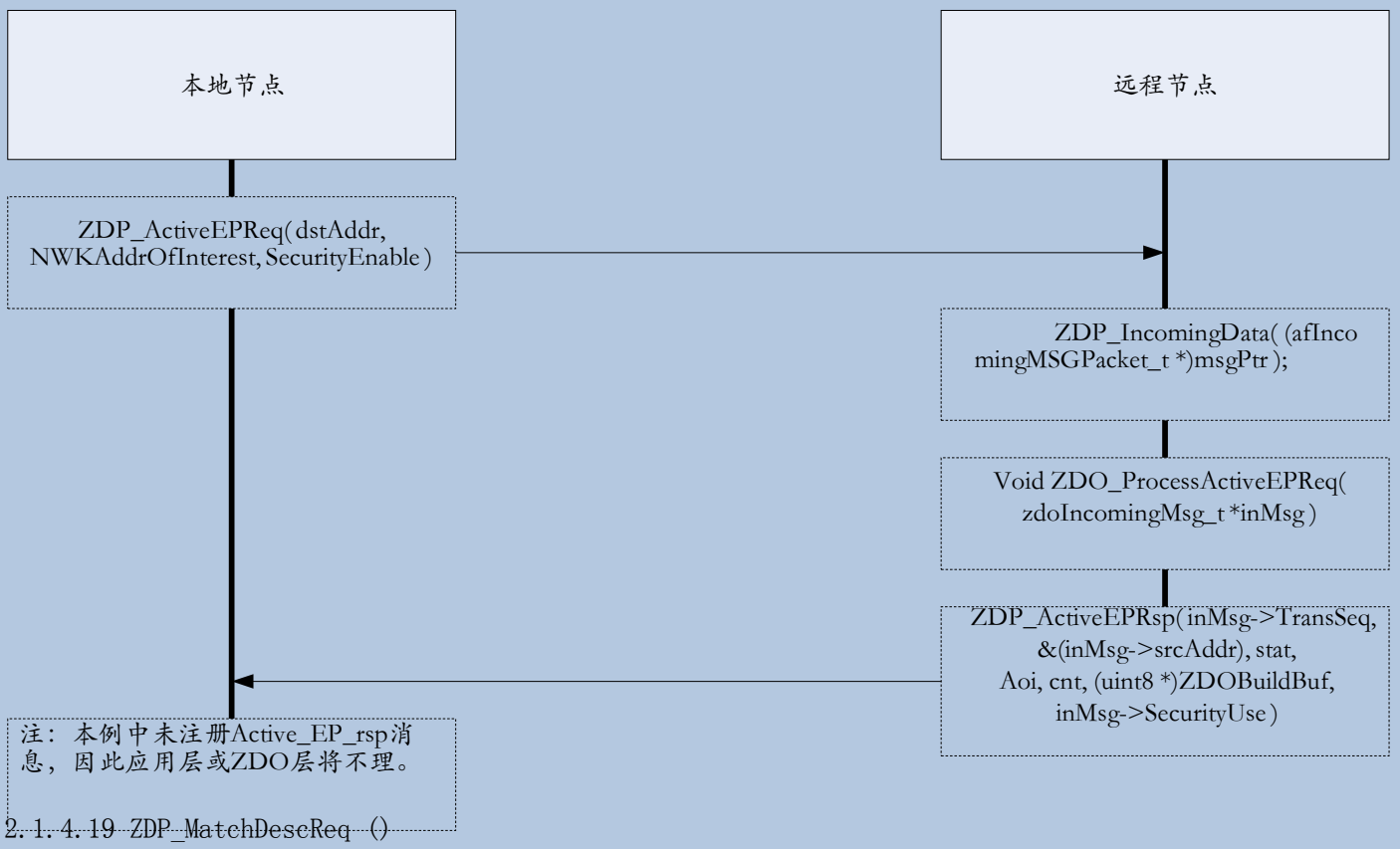
SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

2.1.4.18 活动端点请求和响应应用举例分析

本例中应用层或 ZDO 层均未注册 Active_EP_rsp 信息。调用 ZDP_ActiveEPIFReq ()可以根据已明确了的网络地址请求远程节点的活动端点。远程节点接收到该请求信息（该信息从属于 AF_DATA_CONFIRM_CMD），则根据 Cluster ID 选择处理函数，会使用 zdpProcessActiveEPReq()来处理活动端点请求。当处理完之后，通过调用 ZDP_ActiveEPRsp(inMsg->TransSeq, &(inMsg->srcAddr), stat, aoi, cnt, (uint8 *)ZDOBuildBuf, inMsg->SecurityUse)将响应信息发送至本地节点。详细流程如下图所示:



2.1.4.19 ZDP_MatchDescReq ()

这个函数将建立和发送一个匹配描述符请求。使用这个函数查询与本地节点的输入或输出簇列表

相匹配的远程节点。

函数原型:

afStatus_t ZDP_MatchDescReq(zAddrType_t *dstAddr, uint16 nwkAddr, uint16 ProfileID, byte NumInClusters, byte *InClusterList, byte NumOutClusters, byte *OutClusterList, byte SecuritySuite);

参数:

DstAddr – 目的地址

nwkAddr – 已明确的16位网络地址

ProfileID – 应用规范ID号

NumInClusters – 输入簇的个数

InClusterList – 输入簇列表，每个元素1byte

NumOutClusters –输出簇的个数

OutClusterList –输出簇列表，每个元素1byte

SecuritySuite – 安全要求

返回值:

ZStatus_t-状态

2.1.4.20 ZDP_MatchDescRsp()

该函数实际上是调用宏定义 ZDP_EPIFRsp().调用此函数响应匹配描述符请求。

函数原型:

afStatus_t ZDP_MatchDescRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status, uint16 nwkAddr, byte Count, byte *pEPIntfList, byte SecuritySuite);

参数：

TranSeq-传输序号

DstAddr-目的地址

Status - ZDP_SUCCESS=0， ZDP_DEVICE_NOT_FOUND=1

nwkAddr—已明确的远程节点的16位网络地址

Count —活动端点个数

pEPIntfList—活动端点列表

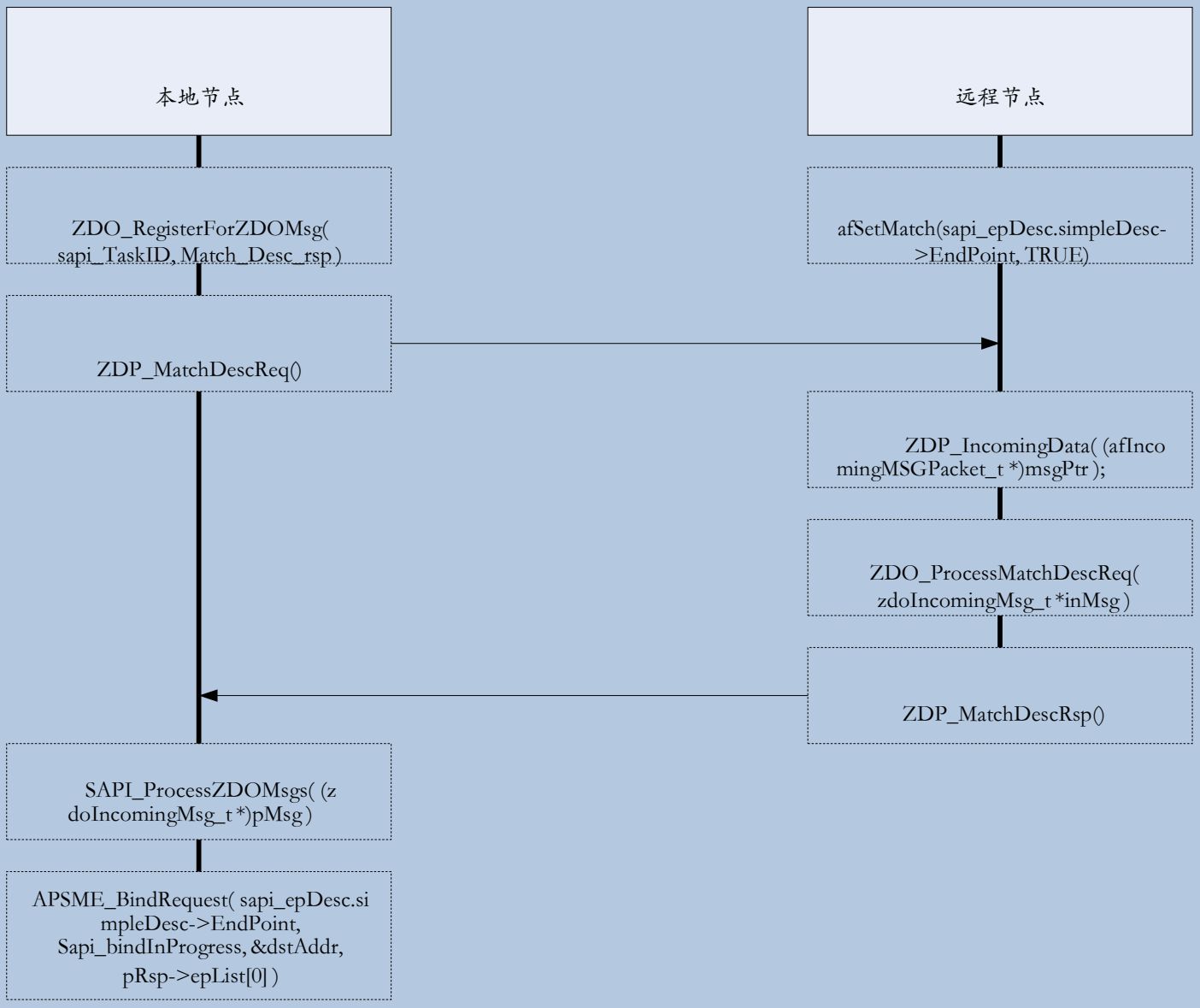
SecuritySuite — 安全要求

返回值：

ZStatus_t-状态

2.1.4.21 匹配描述符请求和响应应用举例分析

本地节点首先需要在应用层或 ZDO 层注册响应信息，本例中在 sapi.c 中注册了 Match_Desc_rsp 信息。另外，远程节点需要开启匹配描述符响应。然后调用 ZDP_MatchDescReq()请求远程节点的匹配描述符。远程节点接收到该请求信息（该信息从属于 AF_DATA_CONFIRM_CMD），则根据 Cluster ID 选择处理函数，本例中使用 ZDO_ProcessMatchDescReq()来处理匹配描述符请求。当处理完之后，通过调用 fillAndSend()将响应信息发送至本地节点。由于在应用层中注册了该响应信息，因此调用 SAPI_ProcessZDOMsgs()来处理响应信息。最后通过调用应用支持子层的函数 APSME_BindRequest()建立绑定表。详细流程如下图所示：



2.1.4.22 ZDP_DeviceAnnce ()

这个函数为 ZigBee 终端设备建立和发送一个 End_Device_annce 命令，来通知网络中的其他 ZigBee 节点。该命令包含终端设备的 16 位网络地址和 64 位 IEEE 地址以及容量。接收到 End_Device_annce 命令的节点将检查 IEEE 地址并更新相应的网络地址，但不会对此命令返回响应信息。

函数原型：

afStatus_t ZDP_DeviceAnnce(uint16 nwkAddr, byte *IEEEAddr, byte capabilities, byte SecurityEnable);

参数：

nwkAddr – 本地节点的16位网络地址

IEEEAddr – 本地节点的64位IEEE地址

Capabilities – 本地节点的容量

SecurityEnable– 安全要求

返回值:

ZStatus_t-状态

2.1.5 ZDO 绑定 API

ZDO 绑定 API 建立和发送 ZDO 绑定请求和响应。所有的绑定表建立在 ZigBee 协调器中。因此，只有 ZigBee 协调器可以接收绑定请求。

ZDO 绑定API	ZDP 绑定服务命令
ZDP_EndDeviceBindReq()	End_Device_Bind_req
ZDP_EndDeviceBindRsp()	End_Device_Bind_rsp
ZDP_BindReq()	Bind_req
ZDP_BindRsp()	Bind_rsp
ZDP_UnbindReq()	Unbind_req
ZDP_UnbindRsp()	Unbind_rsp

2.1.5.1 ZDP_EndDeviceBindReq ()

调用这个函数将建立和发送一个终端设备绑定请求（“手动绑定”）。在手动绑定建立之后，我们可以间接发送信息（无地址）至协调器，协调器将发送此消息至绑定的节点。通过同样的方式，我们也可以接收到绑定节点的信息。

函数原型：

afStatus_t ZDP_EndDeviceBindReq(zAddrType_t *dstAddr, uint16 LocalCoordinator, byte ep, uint16 ProfileID, byte NumInClusters, byte *InClusterList, byte NumOutClusters, byte *OutClusterList, byte SecuritySuite);

参数：

DstAddr – 目的地址

LocalCoordinator – 已知协调器的16位网络地址

ep – 端点

ProfileID – 应用规范ID号

NumInClusters – 输入簇的个数

InClusterList – 输入簇列表，每个元素1byte

NumOutClusters –输出簇的个数 OutClusterList

–输出簇列表，每个元素1byte SecuritySuite–

安全要求

返回值：

ZStatus_t-状态

2.1.5.2 ZDP_EndDeviceBindRsp ()

这个宏定义直接调用 ZDP_SendData()。调用此函数响应终端设备绑定请求。

函数原型:

afStatus_t ZDP_EndDeviceBindRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status, byte SecurityEnable);

参数:

TranSeq – 传输序列号

DstAddr – 目的地址

Status –SUCCESS=0,NOT_SUPPORT=1,TIMEOUT=2,NO_MATCH=3

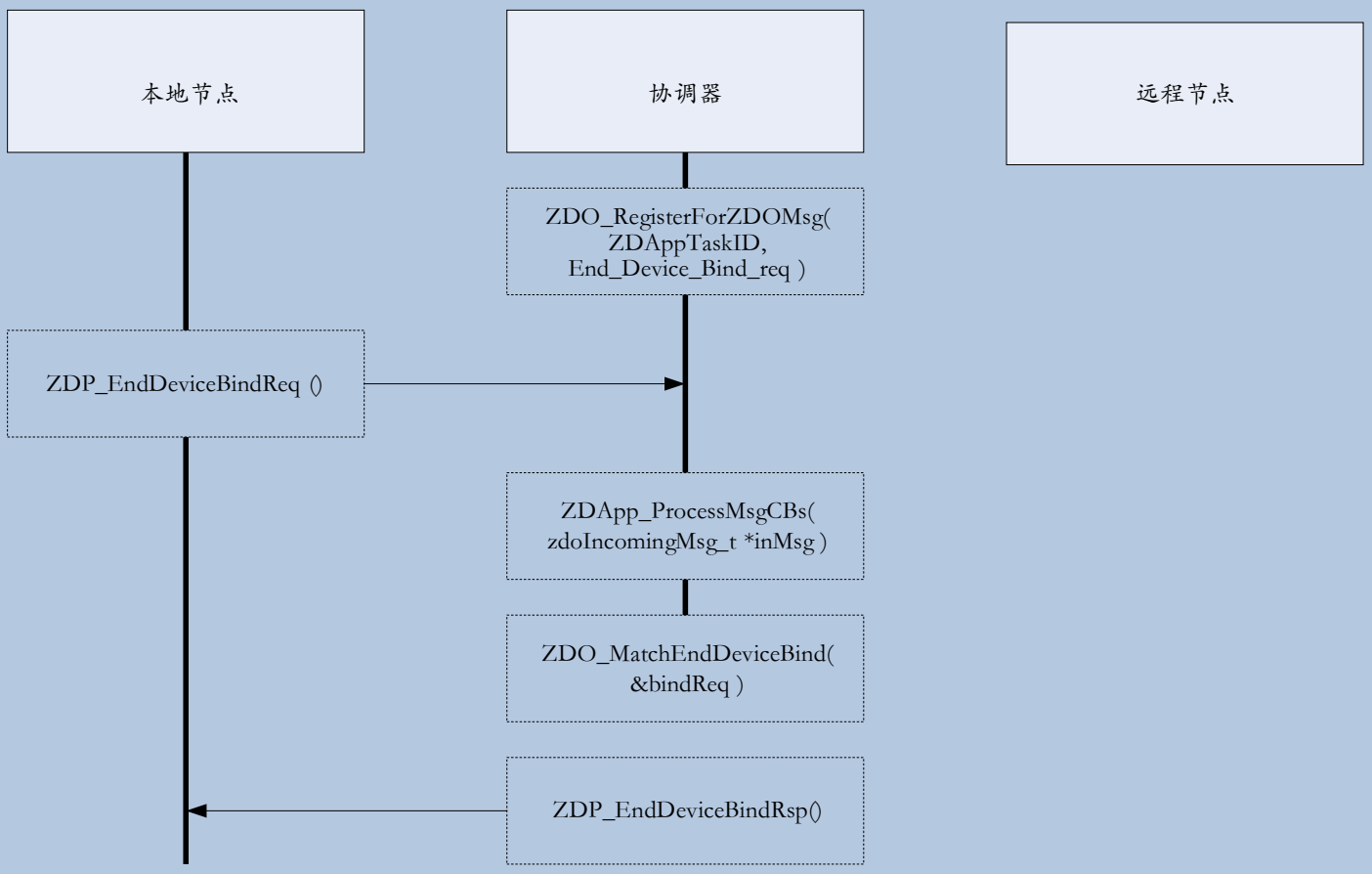
SecuritySuite– 安全要求

返回值:

ZStatus_t-状态

2.1.5.3 终端设备绑定请求和响应应用举例分析

协调器首先需要在ZDO层注册 End_Device_Bind_req信息。然后本地节点调用ZDP_EndDeviceBindReq()发送终端设备绑定请求至协调器。协调器接收到该请求信息（该信息从属于ZDO_CB_MSG），调用ZDO_MatchEndDeviceBind()处理终端设备绑定请求。处理完毕之后，调用ZDP_EndDeviceBindRsp()将反馈信息发送给本地节点。详细流程如下图所示:



2.1.5.4 ZDP_BindReq ()

这个宏定义实际上是调用了函数 ZDP_BindUnbindReq()。该函数将建立和发送一个绑定请求。

函数原型:

afStatus_t ZDP_BindReq(zAddrType_t *dstAddr, byte *SourceAddr, byte SrcEP, byte ClusterID, byte *DestinationAddr, byte DstEP, byte SecuritySuite);

参数:

DstAddr – 目的地址

SourceAddr –本地节点64位IEEE地址

SrcEP – 本地节点的端点

ClusterID – 绑定簇的ID号

DestinationAddr – 远程节点的64位IEEE地址

DstEP – 远程节点的端点

SecuritySuite– 安全要求 返

回值:

ZStatus_t-状态

2.1.5.5 ZDP_BindRsp ()

这个宏定义直接调用 ZDP_SendData()。调用此函数响应绑定请求。

函数原型:

afStatus_t ZDP_EndDeviceBindRsp(byte TranSeq, zAddrType_t *dstAddr, byte Status, byte SecurityEnable);

参数:

TranSeq – 传输序列号

DstAddr – 目的地址

Status –SUCCESS=0,NOT_SUPPORT=1,TABLE_FULL=2

SecuritySuite– 安全要求

返回值:

ZStatus_t-状态

2.1.5.6 绑定请求和响应应用举例分析

本例中协调器欲与远程节点建立绑定连接，协调器在ZDO层注册Bind_rsp消息事件，而远程节点在ZDO层注册Bind_Req消息事件。协调器调用函数ZDP_BindReq()发起绑定请求，远程节点接收到绑

ZDO_CB_MSGZDO_ProcessBindUnbindReq( )

ZDP_SendDataO

ZDO_CB_MSGZDMatchSendState ( )

ZDO_RegisterForZDOMsg(ZDAppTaskIDBind_rsp)

ZDP_BindReqO :

ZDO RegisterForZDOMsg(ZDAppTaskIDBind)

1

: ZDApp_ProcessMsgCBs(zdolncomingMsg_t nMsg)

1 ZDOfmcmBlndunbmdR4 1 inM &bindR)

APSMEBma

zDP Smom()

ZDApp_ProcessMsgCBs(: zdolncomingMsg_t *inMsg) :

ZDMatchSendStateO :

ZDP_BindUnbindReqO

2.2 应用框架（AF）

应用框架层是 APS 层应用程序的无线数据接口。应用程序使用该层提供的功能（通过 APS 层和 NWK 层）来无线收发数据。

2.2.1 概述

AF 提供以下 2 个应用：端点管理和收发数据

2.2.2 端点管理

在 ZigBee 网络中每个设备都是一个节点，每个节点具有唯一的一个 IEEE 地址（64 位）和一个网络地址（16 位）。网络中的其他节点发送数据时必须指定目标节点的短地址，数据才能被接收。每个节点有 241 个端点，其中端点 0 由 ZDO 层使用，它是不可缺少的。端点 1~240 由应用程序分配使用，在 ZigBee 网络中应用程序必须登记注册一个或多个端点，这样才能发送和接收数据。

2.2.2.1 afRegister()

这个函数用来为节点登记注册一个新的端点。应用程序会为每个端点调用这个函数。

函数原型：

afStatus_t afRegister(endPointDesc_t *epDesc);

参数：

epDesc – 端点描述符

返回值：

afStatus_t – 状态

应用举例：

ZDApp.c

```
afRegister( (endPointDesc_t *)&ZDApp_epDesc );
```

其中，ZDApp_epDesc 为端点 0 的端点描述符

```
endPointDesc_t ZDApp_epDesc =
{
  ZDO_EP,
  &ZDAppTaskID,
  (SimpleDescriptionFormat_t *)NULL,  // No Simple description for ZDO
  (afNetworkLatencyReq_t)0            // No Network Latency req
};
```

Sapi.c

```
afRegister( &sapi_epDesc );
```

其中，sapi_epDesc 为应用端点 2 的端点描述符

```
  sapi_epDesc.endPoint = zb_SimpleDesc.EndPoint;
  sapi_epDesc.task_id = &sapi_TaskID;
  sapi_epDesc.simpleDesc = (SimpleDescriptionFormat_t *)&zb_SimpleDesc;
  sapi_epDesc.latencyReq = noLatencyReqs;

const SimpleDescriptionFormat_t zb_SimpleDesc =
{
  MY_ENDPOINT_ID,                 //  Endpoint
  MY_PROFILE_ID,                  //  Profile ID
  DEV_ID_CONTROLLER,              //  Device ID
  DEVICE_VERSION_CONTROLLER,      //  Device Version
  0,                              //  Reserved
  NUM_IN_CMD_CONTROLLER,          //  Number of Input Commands
  (cId_t *) zb_InCmdList,         //  Input Command List
  NUM_OUT_CMD_CONTROLLER,         //  Number of Output Commands
  (cId_t *) NULL                  //  Output Command List
};
```

2.2.2.2 afFindEndPointDesc()

这个函数用来从一个端点查找相应的端点描述符。

函数原型：

```
endPointDesc_t *afFindEndPointDesc( byte endPoint );
```

参数：

endPoint – 端点号

返回值:

endPointDesc_t * – 端点描述符

2.2.2.3 afFindSimpleDesc()

这个函数用来从一个端点查找相应的简单描述符。

函数原型:

byte afFindSimpleDesc(SimpleDescriptionFormat_t **ppDesc, byte EP);

参数:

ppDesc – 简单描述符

EP – 端点号 返回

值: 简单描述符内存

地址

2.2.2.4 afGetMatch()

默认情况下，节点将响应 ZDO 匹配描述符请求，可以使用这个函数获得 ZDO 匹配描述符响应信息的设置。

函数原型:

uint8 afGetMatch(uint8 ep);

参数:

ep – 得到 ZDO 匹配描述符响应信息的端点号

返回值:

true 允许响应, false 不允许或未找到。

2.2.2.4 afSetMatch()

默认情况下，节点将响应 ZDO 匹配描述符请求，可以使用这个函数改变此行为。

函数原型：

uint8 afSetMatch(uint8 ep, uint8 action);

参数：

ep –改变ZDO匹配描述符响应信息行为的端点号

action – true 允许响应（默认值）, false 该端点不允许响应

返回值：

true 成功, false 未找到。

2.2.3 发送数据

2.2.3.1 AF_DataRequest()

调用此函数发送数据。

函数原型：

afStatus_t AF_DataRequest(afAddrType_t *dstAddr, endPointDesc_t *srcEP, uint16 cID, uint16 len, uint8 *buf, uint8 *transID, uint8 options, uint8 radius);

参数：

dstAddr –目的地址。其中地址模式可以是: afAddrNotPresent 绑定模式地址、afAddrGroup组播地址、afAddrBroadcast广播地址、afAddr16Bit直接传输（单播）地址。

srcEP – 源端点

cID – 簇ID号(Cluster ID)

len – buf域的长度，发送数据的字节数。

buf – 准备发送的数据

transID –传输序列号

options – 0x10:AF_ACK_REQUEST 、 0x20:AF_DISCV_ROUTE 、 0x40:AF_EN_SECURITY 、 0x80:AF_SKIP_ROUTING

radius – 最大的跳数 返回

值:

afStatus_t – 状态

2.2.4 接收数据

数据封包发送至登记注册过的端点。应用程序将通过 AF_INCOMING_MSG_CMD OSAL 消息事件处理接收到的数据封包。接收到的数据封包结构如下:

```
typedef struct
{
  osal_event_hdr_t hdr;              /* OSAL Message header */
  uint16 groupId;                    /* Message's group ID - 0 if not set */
  uint16 clusterId;                   /* Message's cluster ID */
  afAddrType_t srcAddr;               /* Source Address*/
  uint16 macDestAddr;                 /* MAC header destination short address */
  uint8 endPoint;                    /* destination endpoint */
  uint8 wasBroadcast;                /* TRUE if network destination was a broadcast address */
  uint8 LinkQuality;                  /* The link quality of the received data frame */
  uint8 correlation;                 /* The raw correlation value of the received data frame */
  int8   rssi;                       /* The received RF power in units dBm */
  uint8 SecurityUse;                  /* deprecated */
  uint32 timestamp;                   /* receipt timestamp from MAC */
  afMSGCommandFormat_t cmd; /* Application Data */
} afIncomingMSGPacket_t;

typedef struct
{
  uint8   event;
  uint8   status;
```

```
} osal_event_hdr_t;


typedef struct
{
  union
  {
    uint16          shortAddr;
    ZLongAddr_t extAddr;
  } addr;
  afAddrMode_t addrMode;
  byte endPoint;
  uint16 panId;   // used for the INTER_PAN feature
} afAddrType_t;

typedef enum
{
  afAddrNotPresent = AddrNotPresent,
  afAddr16Bit          = Addr16Bit,
  afAddr64Bit          = Addr64Bit,
  afAddrGroup          = AddrGroup,
  afAddrBroadcast    = AddrBroadcast
} afAddrMode_t;

typedef struct
{
  byte    TransSeqNumber;
  uint16 DataLength;                      // Number of bytes in TransData
  byte    *Data;
} afMSGCommandFormat_t;
```

2.3 应用支持子层（APS）

APS 层提供以下管理功能：绑定表管理、组表管理、快速地址查询

2.3.1 绑定表管理

绑定表定义在静态 RAM 区中。表的大小由 f8wConfig.cfg 中的 NWK_MAX_BINDING_ENTRIES（表的条目数）和 MAX_BINDING_CLUSTER_IDS（每个条目中簇的个数）决定。该表可以从 nwk_globals.c 中获得。在 APS 层中使用编译指令 REFLFCTOR 启用绑定功能。绑定表结构如下：

```
// Binding Table
BindingEntry_t BindingTable[NWK_MAX_BINDING_ENTRIES];

/* Maximum number of entries in the Binding table. */
-DNWK_MAX_BINDING_ENTRIES=4

typedef struct
{
                              // No src address since the src is always the local device
  uint8 srcEP;
  uint8 dstGroupMode;     // Destination address type; 0 - Normal address index, 1 - Group address
  uint16 dstIdx;              // This field is used in both modes (group and non-group) to save NV and RAM space
                              // dstGroupMode = 0 - Address Manager index
                              // dstGroupMode = 1 - Group Address
  uint8 dstEP;
  uint8 numClusterIds;
  uint16 clusterIdList[MAX_BINDING_CLUSTER_IDS];
} BindingEntry_t;

-DMAX_BINDING_CLUSTER_IDS=4
```

2.3.2 组表管理

组表是一个分配在 RAM[osal_mem_alloc()]中的一个链表，因此随着表中组个数增加，所分配的堆栈也随之增长。表的最大尺寸由定义在 f8wConfig.cfg 中的 APS_MAX_GROUPS 确定。组表可以从 nwk_globals.c 中获得。

```
typedef struct apsGroupItem
```

```
{
  struct apsGroupItem   *next;
  uint8                 endpoint;
  aps_Group_t           group;
} apsGroupItem_t;

typedef struct
{
  uint16 ID;                                    // Unique to this table
  uint8  name[APS_GROUP_NAME_LEN];   // Human readable name of group
} aps_Group_t;
```

2.3.2.1 aps_AddGroup( )

调用此函数可以增加一个组至组表中。

函数原型:

ZStatus_t aps_AddGroup(uint8 endpoint, aps_Group_t *group);

参数:

endpoint – 新增组的端点

group – 增加至组表的组号和组名

返回值:

成功: ZSuccess ， 失败: ZApsDuplicateEntry、ZApsTableFull、ZMemError

2.3.2.2 aps_RemoveGroup( )

调用此函数可以从组表中删除一个组。

函数原型:

ZStatus_t aps_RemoveGroup(uint8 endpoint, uint16 groupID);

参数:

endpoint – 删除组的端点

groupID – 被删除组的组号

返回值:

成功: true ， 失败: false

2.3.2.3 aps_FindGroup( )

调用此函数可以根据端点和组号从组表中查找到一个组。

函数原型:

aps_Group_t *aps_FindGroup(uint8 endpoint, uint16 groupID);

参数:

endpoint – 查找组的端点

groupID – 查找组的组号 返回值: 找

到: 指向组的指针, 未找到: NULL

2.4 网络层（NWK）

网络层为上层提供如下功能: 网络管理、地址管理、网络参数与功能函数。

2.4.1 网络管理

2.4.1.1 NLME_NetworkFormationRequest()

此函数请求组建一个新网络并允许自身成为该网络的 ZigBee 协调器。这个行为的结果返回到 ZDO_NetworkFormationConfirmCB()中。

函数原型:

ZStatus_t NLME_NetworkFormationRequest(uint16 PanId, uint8* ExtendedPANID，uint32 ScanChannels, byte ScanDuration, byte BeaconOrder, byte SuperframeOrder, byte BatteryLifeExtension);

参数:

PanId – 个域网ID号，有效值：0~0xFFFE，如为0xFFFF，则网络层将选择PANID供网络使用。

ExtendedPANID–扩展的个域网ID号

ScanChannels – 扫描的频道，11~26。

ScanDuration – 扫描时间。

BeaconOrder – 该值为： BEACON_ORDER_NO_BEACONS.

SuperframeOrder – 该值为： BEACON_ORDER_NO_BEACONS.

BatteryLifeExtension – 电池寿命延长模式

返回值:

ZStatus_t – 状态

2.4.1.2 网络层-组建网络请求举例分析

调用组建网络请求函数的必要条件是：logicalType == NODETYPE_COORDINATOR 并且 startMode == MODE_HARD。本例中 NLME_NetworkFormationRequest()各参数设置如下:

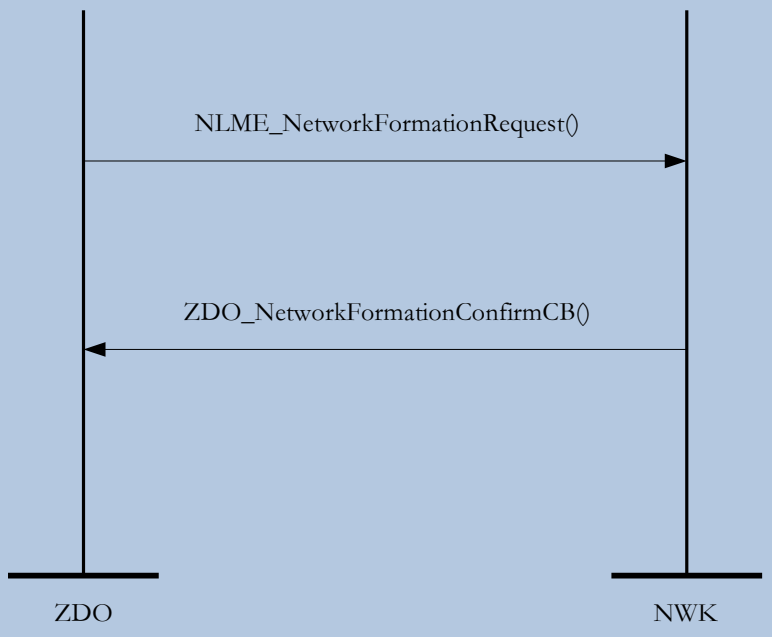
```
f8wConfig.cfg 中定义：-DZDAPP_CONFIG_PAN_ID=0xFFFF
uint16 zgConfigPANID = ZDAPP_CONFIG_PAN_ID;

uint8 zgApsUseExtendedPANID[Z_EXTADDR_LEN] = {00,00,00,00,00,00,00,00};

f8wConfig.cfg 中定义：-DDEFAULT_CHANLIST=0x00000800  // 11 - 0x0B

#define STARTING_SCAN_DURATION          5
uint8 zgDefaultStartingScanDuration = STARTING_SCAN_DURATION;

#define BEACON_ORDER_NO_BEACONS        15
beaconOrder = BEACON_ORDER_NO_BEACONS;
```

```
superFrameOrder = BEACON_ORDER_NO_BEACONS;
```

返回函数 ZDO_NetworkFormationConfirmCB()将设置网络启动事件，交由操作系统做下一步处理。

```
void ZDO_NetworkFormationConfirmCB( ZStatus_t Status )
{
  ……
  osal_set_event( ZDAppTaskID, ZDO_NETWORK_START );
}
```

组建网络请求和反馈发生在协调器的 ZDO 层与 NWK 层之间：



2.4.1.3 NLME_NetworkDiscoveryRequest()

此函数请求网络层寻找相邻的路由器。这个函数应该在加入并执行网络扫描之前调用。这个行为的结果返回到 ZDO_NetworkDiscoveryConfirmCB()中，返回结果包含：发现的路由个数和网络描述列表。

函数原型：

ZStatus_t NLME_NetworkDiscoveryRequest(uint32 ScanChannels, byte ScanDuration);

参数：

ScanChannels – 扫描的频道，11~26。

ScanDuration – 扫描时间。

返回值：

ZStatus_t – 状态

2.4.1.4 网络层-发现网络请求举例分析

调用发现网络请求函数的必要条件是：logicalType == NODETYPE_ROUTER || logicalType == NODETYPE_DEVICE 并且 startMode == MODE_JOIN || startMode == MODE_REJOIN。本例中 NLME_NetworkDiscoveryRequest()各参数设置如下：

```
f8wConfig.cfg 中定义：-DDEFAULT_CHANLIST=0x00000800    // 11 - 0x0B

#define STARTING_SCAN_DURATION          5
uint8 zgDefaultStartingScanDuration = STARTING_SCAN_DURATION;
```

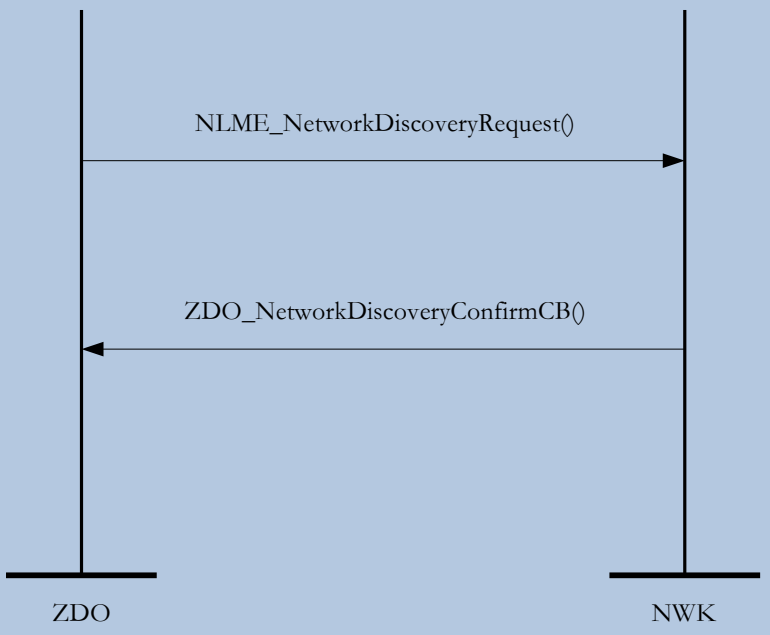
返回函数 ZStatus_t ZDO_NetworkDiscoveryConfirmCB(uint8 ResultCount,networkDesc_t *NetworkList)中网络描述列表定义如下：

```
typedef struct
{
  uint16 panId;
  byte logicalChannel;
  byte beaconOrder;
  byte superFrameOrder;
  byte routerCapacity;
  byte deviceCapacity;
  byte version;
  byte stackProfile;
  uint16 chosenRouter;
  uint8 chosenRouterLinkQuality;
  uint8 chosenRouterDepth;
  uint8 extendedPANID[Z_EXTADDR_LEN];
  byte updateId;
  void *nextDesc;
} networkDesc_t;
```

返回函数 ZStatus_t ZDO_NetworkDiscoveryConfirmCB()将网络列表的部分元素传递至发现网络反馈消息 ZDO_NetworkDiscoveryCfm_t，并设置发现网络消息事件 ZDO_NWK_DISC_CNF，交由 ZDApp 任务事件处理函数做下一步处理：

```
typedef struct
{
   osal_event_hdr_t    hdr;
   uint8            panIdLSB;
   uint8          panIdMSB;
   uint8          logicalChannel;
   uint8          version;
   uint8          extendedPANID[Z_EXTADDR_LEN];
} ZDO_NetworkDiscoveryCfm_t;

ZStatus_t ZDO_NetworkDiscoveryConfirmCB( uint8 ResultCount,networkDesc_t *NetworkList )
{
      networkDesc_t *pNwkDesc = NetworkList;

      msg.hdr.status = ZDO_SUCCESS;
      msg.panIdLSB = LO_UINT16( pNwkDesc->panId );
      msg.panIdMSB = HI_UINT16( pNwkDesc->panId );
      msg.logicalChannel = pNwkDesc->logicalChannel;
      msg.version = pNwkDesc->version;
      osal_cpyExtAddr( msg.extendedPANID, pNwkDesc->extendedPANID );

      ZDApp_SendMsg( ZDAppTaskID, ZDO_NWK_DISC_CNF, sizeof(ZDO_NetworkDiscoveryCfm_t), (uint8
*)&msg );
}
```

发现网络请求和反馈发生在路由器或终端设备的 ZDO 层与 NWK 层之间：



2.4.1.5 NLME_JoinRequest()

此函数请求节点将自己加入到一个网络中。这个行为的结果返回到 ZDO_JoinConfirmCB()中。

函数原型:

ZStatus_t NLME_JoinRequest(uint8 *ExtendedPANID, uint16 PanId,byte Channel, byte CapabilityInfo);

参数:

ExtendedPANID – 试图加入的网络的扩展PAN ID号。

PanId –试图加入的网络的PAN ID号。

Channel –频道号，11~26

CapabilityInfo – 正在加入网络的设备的能力。

返回值:

ZStatus_t – 状态

2.4.1.6 网络层-加入网络请求举例分析

调用加入网络请求函数的必要条件是:devStartMode == MODE_JOIN。本例中 NLME_JoinRequest()各参数设置如下:

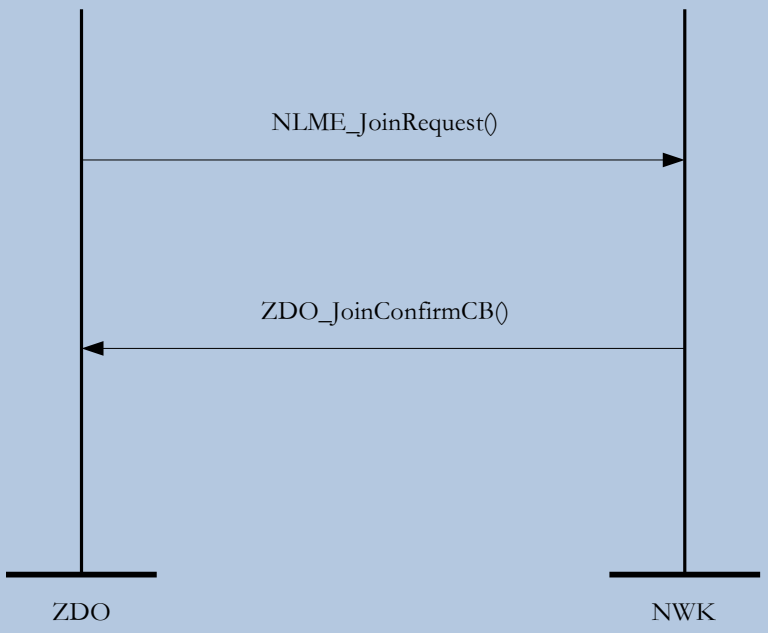
```
(ZDO_NetworkDiscoveryCfm_t *)msgPtr)->extendedPANID: 试图加入的网络的扩展 PAN ID 号
(ZDO_NetworkDiscoveryCfm_t *)msgPtr)->panIdLSB: 试图加入的网络的 PAN ID 号高 8 位
(ZDO_NetworkDiscoveryCfm_t *)msgPtr)->panIdMSB : 试图加入的网络的 PAN ID 号低 8 位
(ZDO_NetworkDiscoveryCfm_t *)msgPtr)->logicalChannel: 频道号
ZDO_Config_Node_Descriptor.CapabilityFlags : 节点描述符中的节点能力
```

返回函数 ZStatus_t ZDO_JoinConfirmCB(uint16 PanId, ZStatus_t Status)设置加入网络消息事件 ZDO_NWK_JOIN_IND，交由 ZDApp 任务事件处理函数做下一步处理:

```
void ZDO_JoinConfirmCB( uint16 PanId, ZStatus_t Status )
{
  ……
  ZDApp_SendMsg( ZDAppTaskID, ZDO_NWK_JOIN_IND, sizeof(osal_event_hdr_t), (byte*)NULL );
```

```
}
```

加入网络请求和反馈发生在路由器或终端设备的 ZDO 层与 NWK 层之间:



2.4.1.7 NLME_ReJoinRequest()

此函数请求节点重新加入到一个已经加入过的网络中。这个行为的结果返回到 ZDO_JoinConfirmCB()中。

函数原型:

ZStatus_t NLME_ReJoinRequest(uint8 *ExtendedPANID, byte Channel);

参数:

ExtendedPANID – 试图加入的网络的扩展PAN ID号。

Channel –频道号, 11~26

返回值:

ZStatus_t – 状态

2.4.1.8 网络层-重新加入网络请求举例分析

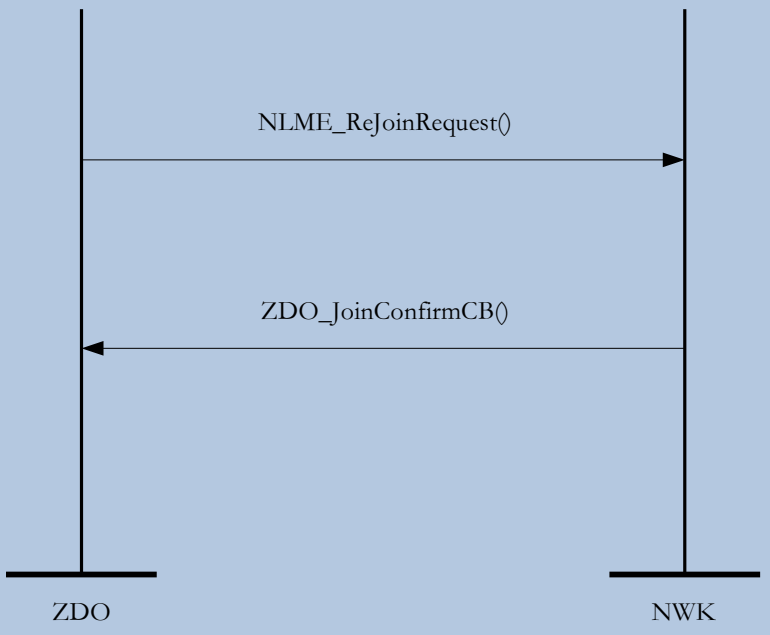
调用重新加入网络请求函数的必要条件是：devStartMode == MODE_ReJOIN。本例中NLME_JoinRequest()各参数设置如下：

```
ZDO_UseExtendedPANID: 试图加入的网络的扩展 PAN ID 号
(ZDO_NetworkDiscoveryCfm_t *)msgPtr)->logicalChannel：频道号
```

返回函数 ZStatus_t ZDO_JoinConfirmCB(uint16 PanId, ZStatus_t Status)设置加入网络消息事件ZDO_NWK_JOIN_IND，交由 ZDApp 任务事件处理函数做下一步处理：

```
void ZDO_JoinConfirmCB( uint16 PanId, ZStatus_t Status )
{
  ……
  ZDApp_SendMsg( ZDAppTaskID, ZDO_NWK_JOIN_IND, sizeof(osal_event_hdr_t), (byte*)NULL );
}
```

重新加入网络请求和反馈发生在路由器或终端设备的 ZDO 层与 NWK 层之间：



2.4.1.9 NLME_OrphanJoinRequest()

此函数请求孤立节点连接到一个父节点。这个行为的结果返回到 ZDO_JoinConfirmCB()中。

函数原型：

ZStatus_t NLME_OrphanJoinRequest(uint32 ScanChannels, byte ScanDuration);

参数：

ScanChannels –频道号，11~26。

ScanDuration –扫描时间 返回值：

ZStatus_t – 状态

2.4.1.10 网络层-孤立节点连接父节点请求举例分析

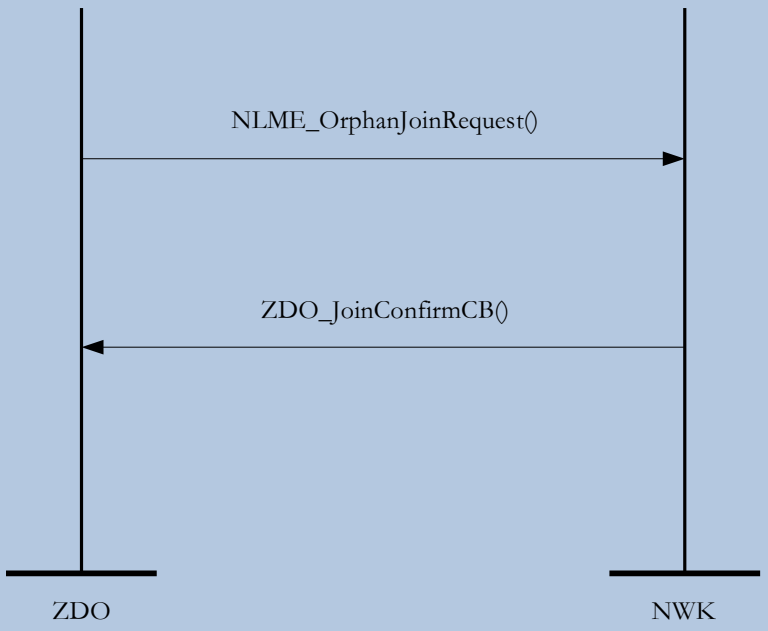
调用孤立节点连接父节点请求函数的必要条件是：startMode == MODE_RESUME 并且 logicalType == NODETYPE_DEVICE。本例中 NLME_OrphanJoinRequest ()各参数设置如下：

```
zgDefaultChannelList = DEFAULT_CHANLIST （频道号=11）
zgDefaultStartingScanDuration = STARTING_SCAN_DURATION（扫描时间=5）
```

返回函数 ZStatus_t ZDO_JoinConfirmCB(uint16 PanId, ZStatus_t Status)设置加入网络消息事件 ZDO_NWK_JOIN_IND，交由 ZDApp 任务事件处理函数做下一步处理：

```
void ZDO_JoinConfirmCB( uint16 PanId, ZStatus_t Status )
{
  ……
  ZDApp_SendMsg( ZDAppTaskID, ZDO_NWK_JOIN_IND, sizeof(osal_event_hdr_t), (byte*)NULL );
}
```

孤立节点连接父节点请求和反馈发生在路由器或终端设备的 ZDO 层与 NWK 层之间：



2.4.1.11 NLME_StartRouterRequest()

此函数请求启动路由器。这个行为的结果返回到 ZDO_StartRouterConfirmCB()中。

函数原型:

ZStatus_t NLME_StartRouterRequest(byte BeaconOrder, byte SuperframeOrder, byte BatteryLifeExtension);

参数:

BeaconOrder –BEACON_ORDER_NO_BEACONS.

SuperframeOrder –BEACON_ORDER_NO_BEACONS.

BatteryLifeExtension –TRUE: 电池供电, FALSE: 电源供电

返回值:

ZStatus_t – 状态

2.4.1.12 网络层-路由器启动请求举例分析

调用路由器启动请求函数的必要条件是: logicalType == NODETYPE_COORDINATOR 并且 startMode == MODE_RESUME, 或者, LogicalType == NODETYPE_ROUTER 并且 devState ==

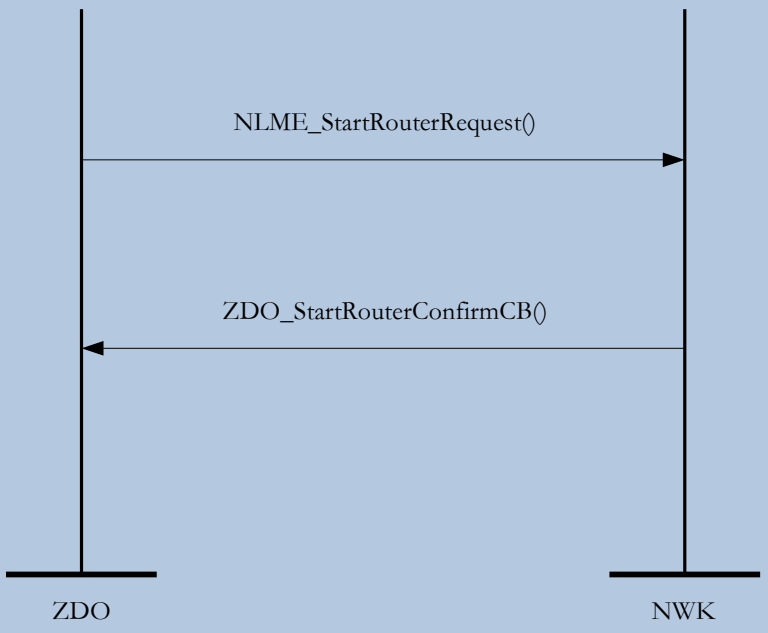
DEV_NWK_JOINING || devState == DEV_NWK_ORPHAN|| devState == DEV_NWK_REJOIN。本例中 NLME_StartRouterRequest()各参数设置如下：

```
BeaconOrder=0
SuperframeOrder=0
BatteryLifeExtension=False
```

返回函数 ZStatus_t ZDO_StartRouterConfirmCB(ZStatus_t Status)设置路由启动事件 ZDO_ROUTER_START，交由 ZDApp 任务事件处理函数做下一步处理：

```
void ZDO_StartRouterConfirmCB( ZStatus_t Status )
{……
    osal_set_event( ZDAppTaskID, ZDO_ROUTER_START );
}
```

路由启动请求和反馈发生在路由器或终端设备的 ZDO 层与 NWK 层之间：



2.4.2 地址管理

查询并存储在本地设备的远程地址。

2.4.2.1 NLME_GetExtAddr()

调用此函数得到节点自身的 64 位 IEEE 地址。

函数原型:

byte *NLME_GetExtAddr(void);

参数: 空

返回值:

指向 64 位 IEEE 地址的指针

2.4.2.2 NLME_GetShortAddr()

调用此函数得到节点自身的 16 位 网络地址。

函数原型:

byte *NLME_GetShortAddr(void);

参数: 空

返回值:

16 位网络地址

2.4.2.3 NLME_GetCoordShortAddr()

调用此函数得到父节点的 16 位 网络地址。

函数原型:

byte *NLME_GetCoordShortAddr(void);

参数: 空

返回值:

父节点 16 位网络地址

2.4.2.4 NLME_GetCoordExtAddr()

调用此函数得到父节点的 64 位 IEEE 地址。

函数原型:

byte *NLME_GetCoordExtAddr(void);

参数: 空

返回值:

指向父节点 64 位 IEEE 地址的指针

2.4.2.5 NLME_IsAddressBroadcast()

此函数根据设备能力来评估提供的地址是否是一个有效的广播地址。

函数原型:

addr_filter_t NLME_IsAddressBroadcast(uint16 shortAddress);

参数:

shortAddress – 被测试的 16 位网络地址

返回值:

addr_filter_t –ADDR_NOT_BCAST 、ADDR_BCAST_FOR_ME 、ADDR_BCAST_NOT_ME

2.4.2.6 NLME_SetBroadCastFilter()

此函数根据设备能力设置掩码，用于处理有效的广播地址

函数原型:

void NLME_SetBroadcastFilter(byte capabilities);

：

capabilities

：